ECE/CS 584: Embedded and cyberphysical system verification Fall 2025

Your name, netid *Homework 1: Modeling computation and invariance*

Due September 12th 11:59pm

Upload your writeup (.pdf) and code (.py) to Gradescope. In the writeup, put each answer on a separate page and label it with the correct number. Ensure your code runs on the python environment described in Problem 4. You may not get points if your code does not run. Code for this homework can be downloaded here.: https://uofi.box.com/s/ugwhbjik45exxbn10ekz9n8g12pwllel

Problem 1. Bidirectional array (20 points, Modeling+Execution+Inductive Invariance Theorem). Model the following token-based mutual exclusion algorithm that works on a *bidirectional array*. There are N processes $\{0, \ldots, N-1\}$ in an array. Each process i, has a single variable $s[i]$ that takes values in the set $\{0, 1, 2, 3\}$ independent of the size of the array. The two processes 0 and $N-1$ behave differently from the rest, they can take two values each $s[0]$ can take values $\{1, 3\}$ and $s[N-1]$ can take values $\{0, 2\}$. Let $Nbrs(i)$ be the set of neighboring processes for process i.

1. Program for processes i, $i = 0$ or $i = N-1$:
   if $\exists\ j \in Nbrs(i)$:$s[j] = s[i] + 1 \mod 4$ then $s[i] = s[i] + 2 \mod 4$
2. Program for processes i, $0 < i < N-1$:
   if $\exists\ j \in Nbrs(i)$:$s[j] = s[i] + 1 \mod 4$ then $s[i] = s[i] + 1 \mod 4$

In this protocol, process i *has token* iff

$$\exists j \in Nbrs(i) : s[j] = s[i] + 1 \mod 4.$$

(a) Write the model of the bidirectional array system with N processes using the language we saw in class.

(b) Write an execution of the algorithm that starts from a state with a single token. Mark the process with the token (a special color / bold font).

(c) Write an execution (of length at least 6) that starts from a state with multiple tokens.

(d) Prove the invariant "system has a single token" using the inductive invariance theorem.

Problem 2. LCR Leader election (30 points, Modeling+Execution). In this problem, you will create a model of a leader election algorithm in a unidirectional ring [2]. Here is the informal description of the protocol:

Each process sends its unique identifier to its successor around the ring. When a process receives an incoming identifier, it compares that identifier to its own. If the incoming identifier is greater than its own, it keeps passing the identifier; if it is less than its own, it discards the incoming identifier and sends out its own identifier; if it is equal to its own the process declares itself as the leader.

(a) Write the model of the system with n processes in the ring using the language we saw in class. To get you started, the set of variables is:

- $send$: The identifier to send or *null*,
- $status$: Takes values in $\{unknown, leader\}$ to indicate that the leader has been elected or not.

(b) Write an execution of the system ($N \geq 4$) in which status of at least one process is eventually set to $leader$.

(c) Is the following statement true: "Suppose the unidirectional ring have N process. Then there have to be at least 1 leader after N+1 transitions."? If yes, prove it; if no, provide a counterexample.

(d) Is the following statement true: "The process with the largest number in the unidirectional ring will be the leader."? If yes, prove it; if no, provide a counterexample.

Problem 3. Impossibility of election (15 points, Model + Inductive Invariance Theorem). A *symmetric function* $f : S^k \rightarrow S$ has the property that for all $s_1, s_2 \in S^k$, if s_1 is a permutation of s_2 then $f(s_1) = f(s_2)$. That is, for $k = 3$, $f(\langle 1, 2, 3\rangle) = f(\langle 2, 3, 1\rangle) = f(\langle 3, 1, 2\rangle)$, etc.

Consider a synchronous algorithm SymGk running on a graph G with *indegree* k. That is, each process P_j reads the states of exactly k other processes. In this algorithm. SymGk, every node updates its state x_i according to some symmetric function $f : S^k \rightarrow S$, that is,

$$x[i] := f(\langle x[j] \mid (j, i) \in G\rangle)$$

Show that it is impossible for SymGk to elect a leader if initially every process has the same value of $x[i]$. *[Hint: State an invariant and prove it by induction on rounds.]*

Problem 4. Dijkstra invariance with Z3 (35 points, Token Ring). Use the Z3 SMT solver to encode and check that for the Dijkstra's token ring mutual exclusion algorithm (DijkstraTR of Chapter 2) the predicate ϕ_{legal} is an inductive invariant.

First, install Z3 on your computer. This is a quick process: on macOS, running `pip install z3-solver` (remark: not z3) usually completes in less than a minute. For Windows and Linux, you can download it directly from https://github.com/Z3Prover/z3. Ample online resources are available to help with installation issues. **Please ensure that your environment matches the test versions used on our side: `python: 3.10` and `z3: 4.8-4.15`.**

Next, download the file `DijkstraTRind.py` provided for this homework and read it carefully. The program contains several functions, some of which you will need to complete. The accompanying documentation, lecture slides, and the discussions in Chapters 2 and 7 of the book should provide sufficient guidance. Additional notes are provided below.

- `has_token(x_list, j)`: Generates a Z3 Boolean expression (predicate) that represents whether process P_j holds the token. Here `x_list` is a list of Z3 variables, for example, `[x[0], x[1], x[2], x[3]]`, and `j` is the index of process P_j.

- `legal_config(x_list)`: Generates a Z3 Boolean expression that represents whether the system is in a legal configuration. This is the implementation of ϕ_{legal}. This function is given to and **you do not have to change it**.
- `transition_relation(old_x_list, new_x_list)`: Generates a Z3 Boolean expression representing transition from `old_x_list` to `new_x_list`.
- `prove(conjecture)`: Checks whether the Boolean predicate `conjecture` is valid using the Z3 solver to check the *unsatisfiability* of the negation of `conjecture`. **You do not have to change this.**

Run your program with `python3 DijkstraTRind.py`. If the functions are written correctly, then validity of the base case and the induction case imply (by the induction invariance Theorem) that `legal_config` is indeed an invariant.

Problem 5. Project activity. Spend a few hours to think, research, and write up a page answering the questions in the Project Jumpstart Page. Use LaTeX for typesetting and BibTeXfor bibliography and references. Collect papers and references to models and tools you find in a bibliography (.bib) file. Although this problem is not graded, you are strongly recommended to turn this in.

Epilogue. Verifying the self-stabilization property of distributed algorithm's like Dijkstra's token ring algorithm and the other examples in this homework is an interesting problem. Several other examples are given in Chapter 17 of [1]. Multi-agent robotic systems, e.g., for formation flight, flocking, and coverage, also possess self-stabilization properties, although they are not explicitly spoken of. Another related interesting problem is to *synthesize* algorithms that are self-stabilizing, given some constraints on communication and variables. Self-stabilization is not an invariant property and requires a different proof technique with *ranking functions*—something we will encounter later in Chapter 10.

References

[1] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach*. Chapman & Hall/CRC, 2nd edition, 2014.

[2] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.